

A FAST BLOCK LOW-RANK DENSE SOLVER WITH APPLICATIONS TO FINITE-ELEMENT MATRICES

AMIRHOSSEIN AMINFAR*, SIVARAM AMBIKASARAN† AND ERIC DARVE‡

1. Abstract. This article presents a fast dense solver for hierarchically off-diagonal low-rank (HODLR) matrices. This solver uses algebraic techniques such as the adaptive cross approximation (ACA) algorithm to construct the low-rank approximation of the off-diagonal matrix blocks. This allows us to apply the solver to any dense matrix that has an off-diagonal low-rank structure without any prior knowledge of the problem. Using this solver, we propose an algorithm to lower the computational cost of the multifrontal sparse solve process for finite-element matrices arising out of 2D elliptic PDEs. This algorithm relies on the fact that dense “frontal” matrices that arise from the sparse elimination process can be efficiently represented as a hierarchically off-diagonal low-rank (HODLR) matrix. We also present an extended review of the literature on fast direct solvers for dense and sparse matrices.

2. Introduction. In many engineering applications, solving large finite element systems is of great significance. Consider the system

$$(2.1) \quad Ax = b$$

arising from the finite element discretization of an elliptic PDE, where $A \in \mathbb{R}^{(N \times N)}$ is a sparse symmetric positive definite matrix. Our goal is to accelerate the direct solve process of such matrices using fast structured dense solvers. The major advantages of direct solvers are that they require no pre-conditioners and are robust in handling ill-conditioned matrices. Further, the accuracy of direct solvers is less sensitive to the eigenvalue distribution in the complex plane. Finally, direct methods preserve any structure present in the problem and can exploit that structure [26].

To be consistent with our previous work, we adopt the notations used in [1]. We should also mention that we use ‘ n ’ to refer to the size of dense matrices and ‘ N ’ to refer to the size of sparse matrices (e.g., number of degrees of freedom in a finite-element mesh).

In the next section, we review the previous literature on both sparse multifrontal solvers and dense structured solvers. We then introduce our black box HODLR solver in section 4. Section 5 discusses the application of the dense HODLR solver to the sparse multifrontal solve process and demonstrates the fast direct solver for a turbine blade geometry. We also show an application in combination with the FETI-DP method [6]. The FETI method is a family of domain decomposition algorithms to accelerate finite-element analysis on parallel computers. We present the results and numerical benchmarks in section 6.

3. Previous Work.

3.1. Dense Matrices. We review previously developed structured dense solvers and discuss them in relation to our solver. Quite a few fast direct solvers have been developed for different dense matrix structures, which have the off-diagonal low-rank property. The HODLR matrix structure is the most general off-diagonal low-rank structure for which fast direct solvers have been developed. Solvers for this matrix class have a computational cost of $\mathcal{O}(n \log^2 n)$. In an HODLR matrix, the off-diagonal low-rank bases do not have a nested structure across different levels. The p-HSS matrix class has a partially nested off-diagonal low-rank structure [1]. Solvers for p-HSS matrices have a computational cost of $\mathcal{O}(n \log n)$ [1]. Finally, solvers for the HSS matrix, which has a full nested off-diagonal low-rank structure, have $\mathcal{O}(n)$ complexity [27, 3].

*MECHANICAL ENGINEERING DEPARTMENT, STANFORD UNIVERSITY

†COURANT INSTITUTE OF MATHEMATICAL SCIENCES, NEW YORK UNIVERSITY

‡MECHANICAL ENGINEERING DEPARTMENT, STANFORD UNIVERSITY

Martinsson and Rokhlin [20] developed an $\mathcal{O}(n)$ direct solver for boundary integral equations using the off-diagonal low-rank property of the arising matrices. Their method is based on the fact that for a matrix of rank r , there exists a well-conditioned column operation which leaves r columns unchanged and sets the remaining columns to zero. Using this idea, they derive a two-sided compressed factorization of the inverse of such an off-diagonal low-rank matrix. Their generic algorithm requires $\mathcal{O}(n^2)$ operations to construct the inverse. However, they accelerate their algorithm to $\mathcal{O}(n \log^\kappa(n))$ when applied to two-dimensional contour integral equations. The implicit assumption in this method is that the matrix has an HSS structure.

Chandrasekaran et al. [4] present a fast $\mathcal{O}(n)$ direct solver for HSS matrices. In their article, they construct an implicit ULV^H factorization of an HSS matrix, where U and V are unitary matrices, L is a lower triangular matrix and H is the transpose conjugate operator. Their method is based on the fact that for a low-rank representation of the form UBV^H , where U and V are thin matrices with r columns, there exist a unitary transformation Q , in which, only the last r rows of QU are nonzero. They use this observation to recursively compress the matrix and solve the linear system of equations. Since this method requires constructing an HSS tree, the authors suggest an algorithm that uses the SVD or the rank revealing QR decomposition recursively to construct the HSS tree in $\mathcal{O}(n^2)$ time.

Gillman et al. [8] proposed an $\mathcal{O}(n)$ algorithm for directly solving integral equations in one-dimensional domains. They apply the Sherman-Morrison-Woodbury formula recursively to an HSS tree structure to achieve $\mathcal{O}(r^2n)$ solve complexity, where r is the rank of the off diagonal blocks in the HSS matrix. They also describe an $\mathcal{O}(r^2n)$ algorithm for constructing an HSS representation of the matrix resulting from a Nyström discretization of a boundary integral equation.

Ho and Greengard [12] present a fast direct solver for HSS matrices. Their method consists of two phases, namely, the precomputation phase consisting of matrix compression and factorization and the solution phase. They use the interpolative decomposition (ID) algorithm to compress the off-diagonal matrix blocks. This algorithm has a computational complexity of $\mathcal{O}(mn \log r + r^2n)$ for a matrix $K \in \mathbb{R}^{m \times n}$. After obtaining the hierarchical matrix compression of the original dense matrix, new variables and equations are introduced into the system of equations. Finally, all equations are assembled into an extended sparse matrix and a conventional sparse solver is used to factorize the sparse matrix. This method has a computational complexity of $\mathcal{O}(n)$ for both the precomputation and solution phases for boundary integral equations in 2D, while in 3D, these phases cost $\mathcal{O}(n^{1.5})$ and $\mathcal{O}(n \log(n))$ respectively.

Kong et al. [17] have developed an $\mathcal{O}(n^2)$ dense solver for hierarchically off-diagonal low-rank (HODLR) matrices. Similar to [20], they accelerate their algorithm to $\mathcal{O}(n \log^2(n))$, when applied to boundary integral equations. Their method uses the Sherman-Morrison-Woodbury formula to construct a one-sided hierarchical factorization of the inverse of these matrices, in which each factor is a block diagonal matrix. The low-rank approximation scheme in their paper is based on the rank revealing QR algorithm. The authors use the pivoted Gram-Schmidt algorithm to obtain r orthogonal basis vectors for the low-rank matrix in question. For a matrix $K \in \mathbb{R}^{m \times n}$ with rank r , this low-rank approximation scheme requires $\mathcal{O}(mnr)$ operations. Then, they use a randomized algorithm from [25] to accelerate their low-rank approximation scheme. This accelerated low-rank approximation algorithm costs $\mathcal{O}(mn \log(l + lnr))$ in the general case where $r < l < \min(m, n)$.

Ambikasaran and Darve [1] present an $\mathcal{O}(n \log^2(n))$ solver for HODLR matrices and an $\mathcal{O}(n \log(n))$ solver for p-HSS matrices. This approach differs from the approach mentioned in [17] in the fact that, while [17] constructs the inverse, [1] constructs a factorization of the matrix. Each factor in this factorization scheme is a block diagonal matrix with each block being a low-rank perturbation of the identity matrix. The authors then use the Sherman-Morrison-Woodbury formula to invert each block in the block diagonal factors. The article uses the Chebyshev low-rank approximation scheme to factorize the off-diagonal blocks.

As mentioned above, solvers for the HSS matrix structure have the lowest computational complexity among other hierarchically off-diagonal low-rank matrix structures. The solvers for HSS matrices have a complexity of $\mathcal{O}(r^2n)$, with r being the rank of approximation. The main problem with the HSS structure is that, generally, constructing the HSS tree is expensive ($\mathcal{O}(n^2)$ for a $n \times n$ matrix [3, 27]). Although HODLR

Article	Matrix Class	Factorization	Application
Martinsson and Rokhlin [20]	HSS	Two sided compressed factorization of the inverse	2D boundary integral equations
Chandrasekaran et al. [4]	HSS	ULV^H factorization of the matrix	Radial basis function matrices
Gillman et al. [8]	HSS	Data sparse factorization of the inverse	1D integral equations with Nyström discretization
Ho and Greengard [12]	HSS	Factorization of the extended sparse system	2D and 3D boundary integral equations
Kong et al. [17]	HODLR	One sided hierarchical factorization of the inverse	Boundary integral equations
Ambikasaran and Darve [1]	HODLR, p-HSS	Block-diagonal factorization of the matrix	Interpolation using radial basis functions
This article	HODLR	Recursive block LU factorization of the matrix	Finite-element matrices

TABLE 3.1
Summary of fast dense structured solvers.

solvers have a complexity of $\mathcal{O}(r^2 n \log^2 n)$, an HODLR tree can be constructed with $\mathcal{O}(rn)$ complexity. That is, the overall complexity (solve + low-rank factorization) of HODLR solvers is $\mathcal{O}(n)$ while the overall complexity of HSS solvers is typically $\mathcal{O}(n^2)$.

Another point worth mentioning is that most of the previously developed solvers use analytical techniques to construct the low-rank approximation of the off-diagonal matrix blocks. Analytic low-rank approximation techniques like the Chebyshev low-rank approximation method are only applicable when the matrix definition involves an analytical kernel function. These techniques cannot be used for a black-box solver where the matrix is known only algebraically (i.e., we only know the numerical values of its entries). Thus, we have to resort to an algebraic low-rank approximation scheme such as the adaptive cross approximation (ACA) algorithm [22] for the low-rank approximation of the off-diagonal blocks.

Finally, in contrast with the mentioned articles that have mostly developed dense solvers for boundary integral equations, we intend to use our dense solver to accelerate the direct solve process of finite-element matrices. Table 3.1 summarizes dense solver algorithms mentioned above.

3.2. Sparse Matrices. As mentioned in section 3.1, we are interested in accelerating the direct solve process for finite-element matrices. In this article, we focus on the finite-element discretization of elliptic PDEs. One common way of factorizing such matrices is using the sparse Cholesky factorization. The efficiency of this algorithm strongly depends on the ordering of mesh nodes [23]. Sparse Cholesky factorization takes $\mathcal{O}(N^2)$ flops with a typical row-wise or column-wise mesh ordering, where N is the number of degrees of freedom [26]. The most efficient method for solving such matrices is the multifrontal method with nested dissection, which takes $\mathcal{O}(N^{1.5})$ flops for two-dimensional meshes [23].

The multifrontal method was originally introduced by Duff & Reid [5] as an extension to the frontal method of Irons [15]. In this algorithm, the overall factorization is done by factorizing smaller dense frontal matrices [18]. For each node or supernode in the elimination tree, the frontal matrix is obtained using an update process called the “extend-add” process which involves updates from the previously eliminated nodes.

There has been some recent efforts to reduce the computational cost of the multifrontal method with nested dissection. Xia et al. [26] observed that frontal and update matrices in the multifrontal elimination process can be approximated with hierarchically semi-separable (HSS) matrices. The authors first develop

a structured extend-add process to facilitate the formation of the frontal matrices using the HSS structure. Next, they perform a structured dense Cholesky factorization on the obtained frontal matrix. The authors use the algorithm in [3] to compute the explicit factorizations of HSS matrices. Using this procedure, they are able to achieve nearly linear time complexity. However, only regular well shaped meshes are considered in the article. Schmitz et al. [23] extend the approach of [26] to a more general setting of unstructured and adaptive grids.

Martinsson [19] uses a spiral elimination approach along with HSS compression of Schur complements to achieve $\mathcal{O}(N \log^2 N)$ time complexity. This approach however, requires a mesh that can be partitioned into concentric annuli.

The approach presented here is based on the multifrontal method [18]. It doesn't require constructing and maintaining HSS trees and can be applied to any mesh structure. Our method is based on the observation that the frontal matrices obtained during the multifrontal elimination process have a hierarchically off-diagonal low-rank (HODLR) structure. This observation was also made by [26]. In order to factorize (eliminate) these frontal matrices, we present a dense HODLR structured solver that scales as $\mathcal{O}(n \log^2 n)$ for an $n \times n$ matrix. In this article, we only demonstrate the structured elimination (solve) process for the frontal matrix corresponding one of the separators in the elimination tree (the top one). We then expect to be able to apply this method to all frontal matrices in the multifrontal elimination process to achieve an overall linear time complexity.

4. A Fast Direct Solver for HODLR Matrices.

4.1. HODLR Matrices. A hierarchically off-diagonal low rank matrix, which we will refer to as an HODLR matrix from here on, is a matrix that has low-rank off-diagonal blocks at multiple levels. As described in [1], a 2-level HODLR matrix, $K \in \mathbb{R}^{n \times n}$, can be written as shown in Eq. (4.1):

$$(4.1) \quad K = \begin{bmatrix} K_1^{(1)} & W_1^{(1)} V_{1,2}^{(1)T} \\ W_2^{(1)} V_{2,1}^{(1)T} & K_2^{(1)} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} K_1^{(2)} & W_1^{(2)} V_{1,2}^{(2)T} \\ W_2^{(2)} V_{2,1}^{(2)T} & K_2^{(2)} \end{bmatrix} & W_1^{(1)} V_{1,2}^{(1)T} \\ W_2^{(1)} V_{2,1}^{(1)T} & \begin{bmatrix} K_3^{(2)} & W_3^{(2)} V_{3,4}^{(2)T} \\ W_4^{(2)} V_{4,3}^{(2)T} & K_4^{(2)} \end{bmatrix} \end{bmatrix}$$

where for a p -level HODLR matrix, $K_i^{(p)} \in \mathbb{R}^{n/2^p \times n/2^p}$, $W_{2i-1}^{(p)}, W_{2i}^{(p)}, V_{2i-1,2i}^{(p)}, V_{2i,2i-1}^{(p)} \in \mathbb{R}^{n/2^p \times r}$ and $r \ll n$. An HODLR structure is the most general representation for a hierarchically off-diagonal low-rank matrix. Further nested compression of the off-diagonal blocks will lead to p-HSS and HSS structures [1].

4.2. Solver Derivation and Algorithm. Consider the following linear equation:

$$(4.2) \quad Kx = F$$

where $K \in \mathbb{R}^{n \times n}$ is an HODLR matrix and $x, F \in \mathbb{R}^{n \times s}$. Now let's write K as a one-level HODLR matrix and rewrite Eq. (4.2) :

$$(4.3) \quad K = \begin{bmatrix} K_1^{(1)} & W_1^{(1)} V_{1,2}^{(1)T} \\ W_2^{(1)} V_{2,1}^{(1)T} & K_2^{(1)} \end{bmatrix} \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \begin{bmatrix} F_1^{(1)} \\ F_2^{(1)} \end{bmatrix}$$

where $x_i^{(1)}, F_i^{(1)} \in \mathbb{R}^{(\frac{n}{2} \times s)}$. We now introduce two new variables $y_1^{(1)}$ and $y_2^{(1)}$:

$$(4.4) \quad y_1^{(1)} = V_{2,1}^{(1)T} x_1^{(1)}$$

$$(4.5) \quad y_2^{(1)} = V_{1,2}^{(1)T} x_2^{(1)}$$

Rearranging (4.3), we have:

$$(4.6) \quad \underbrace{\begin{bmatrix} K_1^{(1)} & 0 & 0 & W_1^{(1)} \\ 0 & K_2^{(1)} & W_2^{(1)} & 0 \\ -V_{2,1}^{(1)T} & 0 & I & 0 \\ 0 & -V_{1,2}^{(1)T} & 0 & I \end{bmatrix}}_{\hat{K}} \underbrace{\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ y_1^{(1)} \\ y_2^{(1)} \end{bmatrix}}_{\hat{x}} = \underbrace{\begin{bmatrix} F_1^{(1)} \\ F_2^{(1)} \\ 0 \\ 0 \end{bmatrix}}_{\hat{F}}$$

At this point, we will basically construct the block LU factorization of \hat{K} . That is, we first eliminate the blocks corresponding to $x_1^{(1)}$ and $x_2^{(1)}$. This requires us to decompose $K_1^{(1)}$ and $K_2^{(1)}$ into their lower and upper diagonal components $(L_{K_1^{(1)}}, U_{K_1^{(1)}})$ and $(L_{K_2^{(1)}}, U_{K_2^{(1)}})$ respectively. Now, we are left with the Schur complement:

$$(4.7) \quad S^{(1)} = \begin{bmatrix} I & V_{2,1}^{(1)T} K_1^{(1)-1} W_1^{(1)} \\ V_{1,2}^{(1)T} K_2^{(1)-1} W_2^{(1)} & I \end{bmatrix}$$

Factorizing the Schur complement into its lower and upper diagonal components, we have:

$$(4.8) \quad \begin{bmatrix} l_1 & 0 \\ l_2 & l_3 \end{bmatrix} \begin{bmatrix} u_1 & u_2 \\ 0 & u_3 \end{bmatrix} = P_S S^{(1)}$$

where P_S is the pivoting matrix associated with the LU factorization of $S^{(1)}$. Putting it all together, we have:

$$(4.9) \quad \begin{bmatrix} L_{K_1^{(1)}} & 0 & 0 & 0 \\ 0 & L_{K_2^{(1)}} & 0 & 0 \\ -\widetilde{V_{2,1}^{(1)T} U_{K_1^{(1)}}^{-1}} & 0 & l_1 & 0 \\ 0 & -\widetilde{V_{1,2}^{(1)T} U_{K_2^{(1)}}^{-1}} & l_2 & l_3 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \zeta_1 \\ \zeta_2 \end{bmatrix} = \begin{bmatrix} F_1^{(1)} \\ F_2^{(1)} \\ 0 \\ 0 \end{bmatrix}$$

$$(4.10) \quad \begin{bmatrix} U_{K_1^{(1)}} & 0 & 0 & L_{K_1^{(1)}}^{-1} W_1^{(1)} \\ 0 & U_{K_2^{(1)}} & L_{K_2^{(1)}}^{-1} W_2^{(1)} & 0 \\ 0 & 0 & u_1 & u_2 \\ 0 & 0 & 0 & u_3 \end{bmatrix} \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ y_1^{(1)} \\ y_2^{(1)} \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ \zeta_1 \\ \zeta_2 \end{bmatrix}$$

where z_1 , z_2 , ζ_1 , and ζ_2 are unknowns associated with solving the lower diagonal equation (forward substitution). As can be seen in the above equations, z_1 and z_2 correspond to the first and second block equations of (4.6) respectively. Hence, they have the same dimensions as $x_1^{(1)}$ and $x_2^{(1)}$. Similarly, ζ_1 and ζ_2 have the same dimensions as $y_1^{(1)}$ and $y_2^{(1)}$ respectively.

Note that we have multiplied the equation blocks corresponding to $y_1^{(1)}$ and $y_2^{(1)}$ by P_S . The tilde on top of $V_{2,1}^{(1)T}$ and $V_{1,2}^{(1)T}$ shows that they are premultiplied by P_S .

The lower diagonal Eq. (4.9) gives:

$$(4.11) \quad l_1 \zeta_1 = \widetilde{V_{2,1}^{(1)}} K_1^{(1)-1} F_1^{(1)}$$

$$(4.12) \quad l_3 \zeta_2 = \widetilde{V_{1,2}^{(1)}} K_2^{(1)-1} F_2^{(1)} - l_2 \zeta_1$$

We can now obtain $y_1^{(1)}$ and $y_2^{(1)}$ using the upper diagonal Eq. (4.10):

$$(4.13) \quad u_3 y_2^{(1)} = \zeta_2$$

$$(4.14) \quad u_1 y_1^{(1)} = \zeta_1 - u_2 y_2^{(1)}$$

At this point, we can write $x_1^{(1)}$ and $x_2^{(1)}$ in terms of $K_1^{(1)-1}$ and $K_2^{(1)-1}$:

$$(4.15) \quad x_1^{(1)} = K_1^{(1)-1} F_1^{(1)} - K_1^{(1)-1} W_1^{(1)} y_2^{(1)}$$

$$(4.16) \quad x_2^{(1)} = K_2^{(1)-1} F_2^{(1)} - K_2^{(1)-1} W_2^{(1)} y_1^{(1)}$$

Since, both $K_1^{(1)}$ and $K_2^{(1)}$ are HODLR matrices, we can apply the same procedure for solving them. Thus, we have arrived at a recursive algorithm for solving (4.6). The L and U factors of the original matrix are not constructed by instead we calculate these factors for the extended matrix Eq. (4.6). The factorization step corresponds to the computation and storage of all the terms that are independent of the right hand side (i.e., the Schur complements at all levels).

4.2.1. Algorithm Summary. As mentioned before, our goal is to setup a recursive solver. For a matrix such as $K \in \mathbb{R}^{n \times n}$, we have to carry out the following procedure at each recursion step (p) for all $1 \leq i \leq 2^p$:

1. Find the low-rank approximation of the off-diagonal blocks ($W_{2i-1}^{(p)}$, $W_{2i}^{(p)}$, $V_{2i-1,2i}^{(p)}$, $V_{2i,2i-1}^{(p)}$).
2. Recursively solve the following equations with multiple right hand sides:

$$(4.17) \quad \begin{bmatrix} d_{2i-1}^{(p)} & c_{2i-1}^{(p)} \end{bmatrix} = K_{2i-1}^{(p)-1} \begin{bmatrix} W_{2i-1}^{(p)} & F_{2i-1}^{(p)} \end{bmatrix}$$

$$(4.18) \quad \begin{bmatrix} d_{2i}^{(p)} & c_{2i}^{(p)} \end{bmatrix} = K_{2i}^{(p)-1} \begin{bmatrix} W_{2i}^{(p)} & F_{2i}^{(p)} \end{bmatrix}$$

3. Obtain $S^{(p)}$ using Eq. (4.7) and decompose it into its lower and upper diagonal components to obtain $l_{j,i}^{(p)}$ and $u_{j,i}^{(p)}$ ($j \in \{1, 2, 3\}$) and $P_S^{(p)}$.
4. Obtain $y_{2i-1}^{(p)}$, $y_{2i}^{(p)}$ using Eqs. (4.11), (4.12), (4.13), and (4.14).
5. Obtain $x_{2i-1}^{(p)}$, $x_{2i}^{(p)}$ using Eqs. (4.15) and (4.16).

4.3. Solver Computational Cost. In this section, we compute the computational cost of our dense solver algorithm. We first assume that we use a fast low-rank approximation scheme like the Chebyshev low-rank approximation [1], [7] or the ACA low-rank approximation scheme [22]. Using such algorithms, the cost of constructing and storing an HODLR matrix is $\mathcal{O}(nr \log(n))$ [1], where r is the rank of approximation. Further, we assume that the rank of approximation at all levels of the HODLR matrix is equal to r . Looking at the procedure described in section 4.2.1, we can write the following:

$$(4.19) \quad C^{(p)}(r, s, n) = 2C^{(p+1)}(r, s + r, \frac{n}{2}) + \mathcal{O}(nr^2 + nsr)$$

where $C^{(p)}(r, s, n)$ is the computational cost associated with solving an $n \times n$ HODLR matrix at level p with s right hand sides and off-diagonal blocks of rank r . Eq. (4.19) suggests that the cost of solving an HODLR matrix at level p with s right hand sides is made up of two contributions. The first contribution is associated with solving the two diagonal blocks at the lower level ($p + 1$) with $s + r$ right hand sides. The second contribution comes from constructing the update matrix $S^{(p)}$ [Eq. (4.7)]. Writing Eq. (4.19) as a sum, we have:

$$(4.20) \quad C^{(0)}(r, s, n) = \sum_{p=1}^{\log(\frac{n}{r})} n(r^2 + prs) = \mathcal{O}(r^2 n \log^2(n))$$

4.4. Low-Rank Approximation Schemes. In this section we discuss the various low-rank approximations schemes we used for obtaining a low-rank representation of the off-diagonal blocks of our HODLR matrices.

4.4.1. Chebyshev Low-Rank Approximation. The Chebyshev low-rank approximation scheme is only presented here for comparison purposes. For detailed explanations of this algorithm, see for example [7] and [1]. One of the main differences between this algorithm and the ACA algorithm is that the rank of approximation is user-determined and fixed among various levels in the hierarchy. Hence, we can verify the convergence of our solve algorithm by plotting the solver error vs. rank of approximation.

4.4.2. ACA Low-Rank Approximation. As mentioned before, we use the ACA algorithm with partial pivoting as described by Sergej Rjasanow [22] as our low-rank approximation algorithm. This algorithm is an algebraic low-rank approximation scheme and works on any dense matrix without any prior knowledge of the matrix. Further, the rank of approximation in this algorithm is dynamically determined at each level of solve, which, as we will show, will result in an increase in solver speed. Finally, as we will see, this method is capable of handling kernels with singularity on the diagonal. The ACA algorithm has a cost of $\mathcal{O}(r(m+n))$, for a matrix $A \in \mathbb{R}^{m \times n}$ [22], where r is the rank of approximation.

One important thing to note about the ACA algorithm is the stopping criterion. We use the same stopping criterion mentioned in [22]. As suggested by Rjasanow [22], a good stopping criterion is when the following holds for iteration $k = r$:

$$(4.21) \quad \|w_r\|_F \|v_r\|_F \leq \epsilon_{ACA} \|S_r\|_F$$

where w_r and v_r are the r th low-rank approximate matrices, S_r is the approximate matrix at the r th iteration, and ϵ_{ACA} is an arbitrary tolerance chosen by the user.

4.4.3. Pseudo-skeleton Low-Rank Approximation. We have also used the pseudo-skeleton low-rank approximation scheme in our solver. This is mainly because, sometimes, the ACA algorithm does not give accurate results in cases where interacting clusters have points that are very close (e.g., the interaction between two halves of a closed curve). The pseudo-skeleton algorithm allows us to choose points more judiciously in the regions that are close to each other, and thus achieve a higher low-rank approximation accuracy.

As mentioned in [9], for a low-rank matrix A , if we pick a set of row indices ($i \in I = \{i_1, \dots, i_r\}$) and a set of column indices ($j \in J = \{j_1, \dots, j_r\}$) and define matrices C and R such that :

$$(4.22) \quad R = A(I, :)$$

$$(4.23) \quad C = A(:, J)$$

Then, we can approximate A to be :

$$(4.24) \quad A \approx C \hat{A}^{-1} R$$

where $\hat{A} = A(I, J)$. In some cases, the Moore-Penrose pseudo-inverse is needed for \hat{A}^{-1} .

In the case of points along curves, when the points in one cluster may be very close to points in the other cluster, we need to select more points near the boundary points of the curves in order to maintain accuracy and efficiency. For this we constructed a function that, given a set of indices $[i_a, i_b]$ and a desired number of indices r , outputs a set of indices $[i_1, i_r]$ such that a sufficient number of indices in the output sequence are located near i_a and i_b . The ‘‘Chebyshev’’ interpolation nodes provide such a functionality. Given an interval $[a, b]$, the usual Chebyshev nodes in $[a, b]$ are located near a and b . Hence, in order to select the set of row indices I , we consider the interval $[i_1, i_n]$. We then calculate the usual Chebyshev nodes in this interval, which are not integers. For each Chebyshev node x_k , we set the corresponding row index, i_k , to be the closest integer to x_k . The sequence of column indices is found in a similar manner. Although a heuristic, this approach has turned out to work remarkably well.

To monitor the error in the scheme, we pick rows and columns that are not in the set of rows and columns already chosen for low-rank approximation. We then monitor the relative Frobenius norm error on these rows and columns and increase the rank of the approximation until the relative Frobenius norm error falls below a certain tolerance.

For a rank r pseudo-skeleton low-rank approximation, the inversion of \hat{A} has a computational cost of $\mathcal{O}(r^3)$. Monitoring the error has a computational cost of $\mathcal{O}(mr + nr - r^2)$ for $A \in \mathbb{R}^{m \times n}$. Thus, this method has an asymptotic complexity of $\mathcal{O}(nr)$.

5. Application for Multifrontal Solve Process. In this section, we will demonstrate how our fast dense solver algorithm can be applied to a sparse multifrontal solve process. We will not explain the multifrontal algorithm in detail. For a detailed review of the multifrontal method see [18]. We will show how we apply our dense HODLR solver to two practical engineering problems, namely the heat transfer equation in a turbine blade geometry and a FETI-DP solver for a three dimensional elasticity problem. The results will show that our dense solver is applicable to complicated mesh structures in 2D and regular mesh structures in 3D. Our ultimate goal is to apply our fast dense solver to the dense “frontal” matrices obtained in the multifrontal elimination process of a sparse finite-element matrix, and speed up the algorithm to $\mathcal{O}(N)$. The results shown in this paper can be viewed as a proof of concept of this idea.

5.1. Heat Transfer Problem in a Turbine Blade Geometry.

5.1.1. Background. Today, gas turbines are used for aircraft propulsion, land-based power generation and other industrial applications. The thermal efficiency and power output of a gas turbine increases with the turbine rotor inlet temperature [10]. Modern gas turbines are designed to run at high temperatures well in excess of current metal temperature limits [10]. It is very important to accurately predict the temperature distribution in a turbine blade as the blade life may be reduced by half if the prediction is only off by 50° F [10]. Further, an accurate prediction of the turbine blade temperature distribution can help in designing an efficient cooling system and avoiding local hot-spot overheating [10].

The heat transfer problem in a turbine blade is a conjugate heat transfer problem. That is, it is a coupled convection and conduction heat transfer problem. One popular way of obtaining the turbine temperature distribution is to first guess the convective heat transfer coefficient. Next, solve the heat conduction problem using the guessed boundary conditions in the turbine blade geometry. Now that the temperature distribution has been obtained for the boundary surfaces, one can go back and correct the initial guess for the convection heat transfer coefficient. This process is then repeated until the turbine blade temperature distribution converges. This process requires us to solve the conduction problem (heat equation) several times. One way of solving the conduction problem is to use the finite element method to discretize the heat equation on the blade geometry. Hence, a fast and accurate direct solver can significantly speed up the solve process of this problem.

We considered the sample turbine blade geometry shown in Figure 5.1 as described in [24]. This particular blade geometry is similar to the stage-1 high pressure turbine blade design of the NASA E^3 engine [10].

5.1.2. Obtaining the Schur Complement. Consider the heat equation with Neumann boundary conditions:

$$(5.1) \quad \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \quad \text{in } \Omega$$

$$(5.2) \quad \frac{\partial u}{\partial n} = f_i(s), \quad \text{in } \partial\Omega_i$$

We used Freefem++ [11] to obtain the stiffness matrix corresponding to the finite-element discretization of the above equation on our turbine blade geometry. As the stiffness matrix is the same regardless of the boundary condition, we used Dirichlet boundary conditions, as the rows and columns corresponding to boundary nodes are more easily identifiable in this case. We then reorder the rows and columns of the sparse

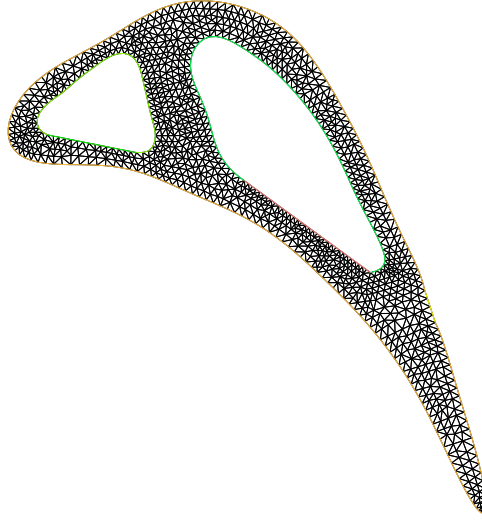


FIG. 5.1. A sample finite-element mesh for the turbine blade geometry

stiffness matrix such that the rows and columns corresponding to the nodes on the boundaries of the mesh ($\partial\Omega_i$) are located at the bottom right corner of the matrix. Further, the ordering is such that the rows and columns corresponding to adjacent boundary nodes are grouped together in the matrix. Now, we eliminate the unknowns corresponding to the nodes lying inside the turbine blade:

$$(5.3) \quad A = \begin{bmatrix} B & V^t \\ V & C \end{bmatrix} = \begin{bmatrix} L_B & 0 \\ VL_B^{-t} & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & C - VB^{-1}V^t \end{bmatrix} \begin{bmatrix} L_B^t & L_B^{-1}V^t \\ 0 & I \end{bmatrix}$$

Here, the matrix block B corresponds to self interaction of all the inner nodes, the blocks V , V^t correspond to the interaction of inner nodes with the boundary nodes, and block C corresponds to the self-interaction of the boundary nodes. As can be seen in Eq. (5.3), the part of the matrix that remains to be factored is the Schur complement, $C - VB^{-1}V^t$. The main point is that this Schur complement is dense and has a certain structure, namely the hierarchically off-diagonal low-rank (HODLR) structure. Figure 5.2 shows this structure for a Schur complement obtained from the elimination of a mesh with approximately 640k total nodes and 10k nodes on the boundaries.

The Schur complement structure clearly shows that the off-diagonal blocks are in fact low-rank and thus, we can use our black-box dense solvers for HODLR matrices to factorize this matrix.

5.1.3. Solving the HODLR Schur Complement. In order to use our HODLR solver for solving the obtained Schur complement, we need to accurately approximate the off-diagonal matrices at each level of the HODLR matrix as a low-rank product. As mentioned in section 4.4.3, our experiments with the ACA low-rank approximation scheme show that this method does not yield accurate results in cases where interacting intervals coincide in more than one point (e.g., the interaction between two halves of a closed curve). This is the case for the off-diagonal blocks corresponding to the self interaction of the points lying on the boundaries of the mesh at the top level. In such cases, rows and columns corresponding to points near the boundaries of the two intervals contain most of the matrix information. The partial pivoting ACA scheme fails to identify these important rows and columns. Thus, we resort to the pseudo-skeleton low-rank approximation scheme to approximate the matrices corresponding to the interaction of such intervals.

The pseudo-skeleton scheme, while being an $\mathcal{O}(rn)$ scheme, allows us to manually select rows and columns out of our matrix. As we anticipate most of the interaction to be near the boundaries of the two intervals,

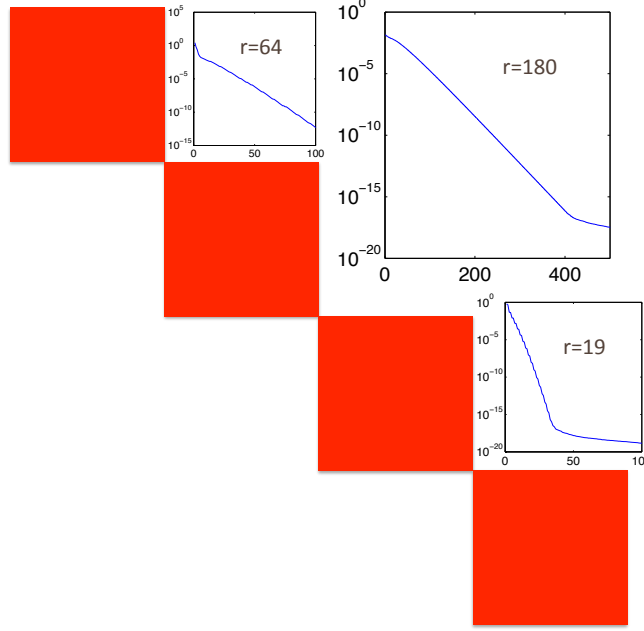


FIG. 5.2. *HODLR structure of the root separator Schur complement of size $\approx 10k \times 10k$. This figure shows the singular value decay of the off-diagonal blocks, where r is the rank of approximation.*

we use “Chebyshev” nodes to choose the corresponding rows and columns. That is, for matrix of size $n \times n$, we consider the Chebyshev nodes in the interval $[0, n]$ and round them to nearest integer to find the index of interest. As Chebyshev nodes mostly lie close to the interval boundaries, this heuristic produced accurate and efficient low-rank approximations.

5.2. FETI-DP Solver for a 3D Elasticity Problem. Domain decomposition (DD) methods solve a problem by splitting it into several subdomains. Local problems are solved on each subdomain and a global linear system is used to couple these local solutions into a global solution for the entire problem [2]. FETI methods are a family of domain decomposition algorithms with Lagrange multipliers that have been developed for the fast sequential and parallel iterative solution of large-scale systems of equations arising from the finite-element discretization of partial differential equations [6].

In this article, we consider a sparse local FETI-DP matrix arising from the finite-element discretization of an elasticity problem in three dimensions. The sparse matrix we are considering has a dimension of $100k \times 100k$. The matrix corresponds to the stiffness matrix of one subdomain of a linear elastic 3D solid finite element model of a cube. The discretization uses 8-node (trilinear) hexahedral elements (see for example section 11.3 of this online document¹).

The first step of most direct solve algorithms for a sparse matrix is to reorder its rows and columns to achieve an ordering that introduces minimal fill-in and create what is called the elimination tree. In our case, we used SCOTCH [21] to do the reordering of the matrix. Similar to Section 5.1.2, we only create a two-level elimination tree. That is, we only consider the root separator and the Schur complement that is created after eliminating the other degrees of freedom. In our case, the Schur complement has a size of $2,500 \times 2,500$.

Next, we use SCOTCH to recursively partition the graph corresponding to the root separator. This step is required in order to permute the obtained Schur complement such that it has an HODLR structure.

¹<http://www.colorado.edu/engineering/CAS/courses.d/AFEM.d/AFEM.Ch11.d/AFEM.Ch11.pdf>

Finally, we apply the black-box HODLR solver to the permuted Schur complement.

This whole procedure was therefore conducted in a “black-box” manner without any direct knowledge of where the matrix is coming from (mesh, vertices, elements, etc), and using only the sparsity pattern of the matrix and the general-purpose graph partitioner SCOTCH.

6. Numerical Benchmarks.

6.1. Dense HODLR Solver Benchmarks. In this section we will show some numerical results as well as simple benchmarks of our code.

6.1.1. HODLR Solver with Chebyshev Low-Rank Approximation. As mentioned in Section 4.4.1, we only use the Chebyshev low-rank approximation for comparison and verification purposes. Figure 6.1(a) shows the convergence plots for various radial basis kernels. The matrices correspond to a uniform point distribution in 1D. As seen in the plots, the solver relative error converges as the rank of approximation increases.

6.1.2. HODLR Solver with ACA Low-Rank Approximation. To benchmark the accuracy of our ACA HODLR solver, we test the code with the dense matrices arising from radial basis functions. Similar to Section 6.1.1, the matrices correspond to a uniform point distribution on a one dimensions interval. The solver convergence plots are shown in Figure 6.1(b).

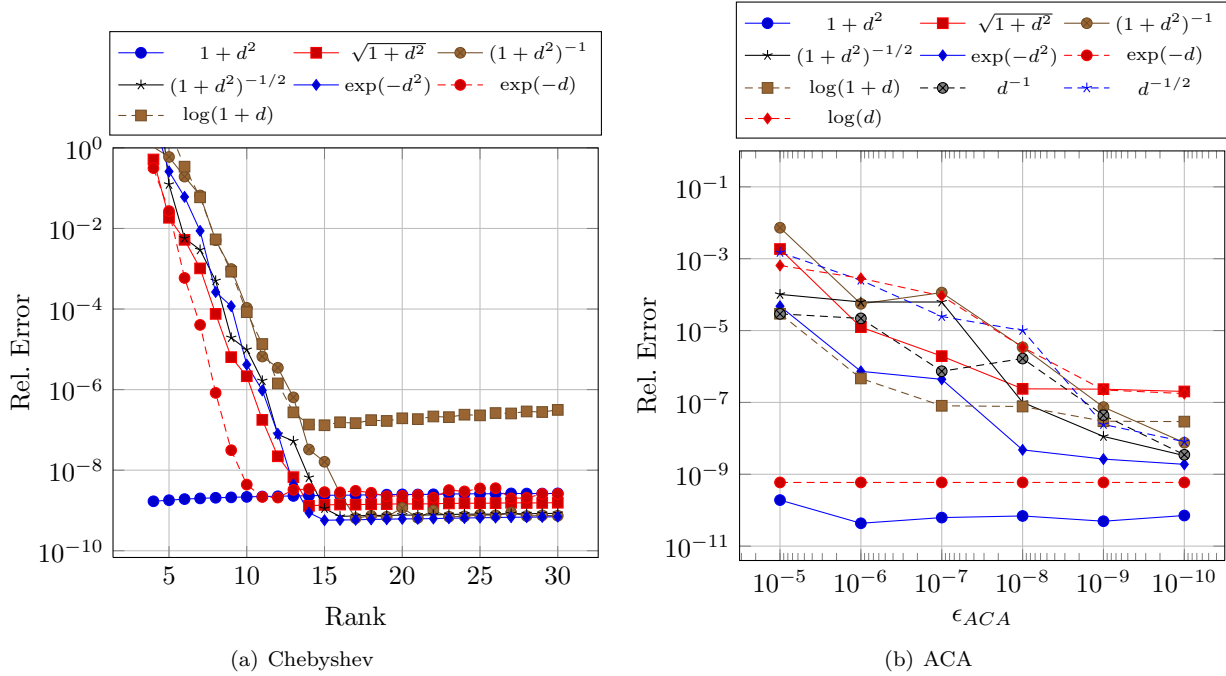


FIG. 6.1. Convergence plots for the solver using Chebyshev and ACA low-rank approximation schemes. These plots show the relative error in the solver for matrices with various radial basis function kernels and a size of $10,000 \times 10,000$. The matrices correspond to a uniform point distribution on the interval $[-1, 1]$. The diagonal entries of all the matrices are set to 0. In the legend, d denotes the distance between the points in $[-1, 1]$. See Eq. (4.21) for the definition of the stopping criterion ϵ_{ACA} .

As can be seen, our code is quite accurate for various radial basis kernels. Moreover, as Figure 6.1(b) shows, the ACA solver is capable of handling kernels with singularity on the diagonal ($d = 0$). This is one the unique advantages of the ACA algorithm compared to the Chebyshev low-rank approximation scheme.

$1 + d^2$	$\sqrt{1 + d^2}$	$(1 + d^2)^{-1}$	$(1 + d^2)^{-1/2}$	$\exp(-d^2)$	$\exp(-d)$	$\log(1 + d)$	d^{-1}	$d^{-1/2}$	$\log(d)$
1.7	1.1	58	5.8	1.5	54	4,800	76	250	310

TABLE 6.1

Condition numbers $\times 10^{-4}$ for 1D radial basis function matrices of size $10,000 \times 10,000$. The relatively high condition numbers (e.g., $5 \cdot 10^7$ for $\log(1 + d)$) means that it is difficult to have roundoff errors smaller than about 10^{-9} , unless an iterative refinement or other scheme is used to reduce the error.

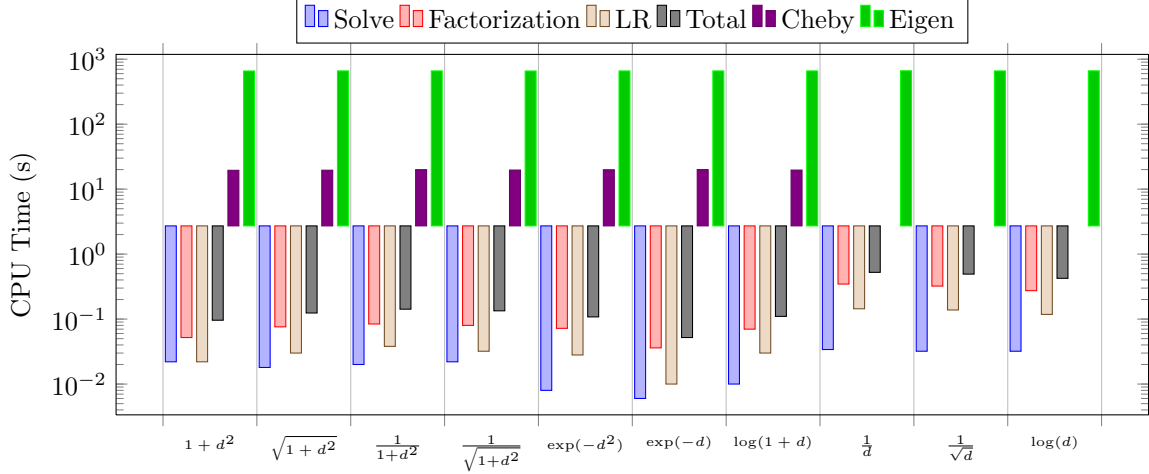


FIG. 6.2. CPU time for various solve stages of the HODLR solver with ACA low-rank approximation for matrices with various radial basis function kernels and a size of $10,000 \times 10,000$. The matrices correspond to a 1D uniform point distribution on the interval $[-1, 1]$. The diagonal entries of all the matrices are set to 0. The ACA results have been obtained with $\epsilon_{ACA} = 10^{-10}$. The CPU time for the HODLR solver with the Chebyshev low-rank approximation and the Eigen colQR [16] solver (QR decomposition with column pivoting) have also been included for comparison purposes. LR is the cost of computing the low-rank approximation of the blocks for the HODLR format using ACA. The last few kernels do not have timings for the Chebyshev method since the kernels are singular.

We can also observe that as we decrease the ACA tolerance (ϵ_{ACA} , see Eq. (4.21)), we arrive at lower errors in our final solution. Table 6.1 shows the condition numbers for the benchmark matrices.

6.1.3. HODLR Solver Speed Benchmarks. In this section we will compare solver speed for various algorithms. Figures 6.2 shows the results of such comparison. We have compared our fast solver to the QR solver in the Eigen C++ library [16] as a standard dense solver. As shown in the figures, for non singular kernels, both the Chebyshev fast solver and the ACA fast solver perform much better than a conventional dense solver. As can be seen in Fig. 6.1(b), the solver is very accurate for $\epsilon_{ACA} = 10^{-10}$, which is the tolerance for which the speed results are obtained. Thus, the relative speed up does not result in a significant loss of accuracy.

One can observe that the ACA solver is faster than both the Chebyshev and Eigen solvers. Fig. 6.2 shows the factorization time, low-rank approximation time, solve time and total solve time separately for the ACA solver.

Figure 6.3 shows the CPU time vs matrix size (n). As can be seen both the fast solver with Chebyshev low-rank approximation and the fast solver with ACA low-rank approximation scale with $\mathcal{O}(n \log^2(n))$.

6.2. Turbine Blade Schur Complement Results. Figure 6.4(a) shows the accuracy of our dense solvers when solving Schur complements obtained from a one level elimination of the turbine blade stiffness matrix. As the accuracy plots show, the relative error decreases with tightening the low-rank approximation

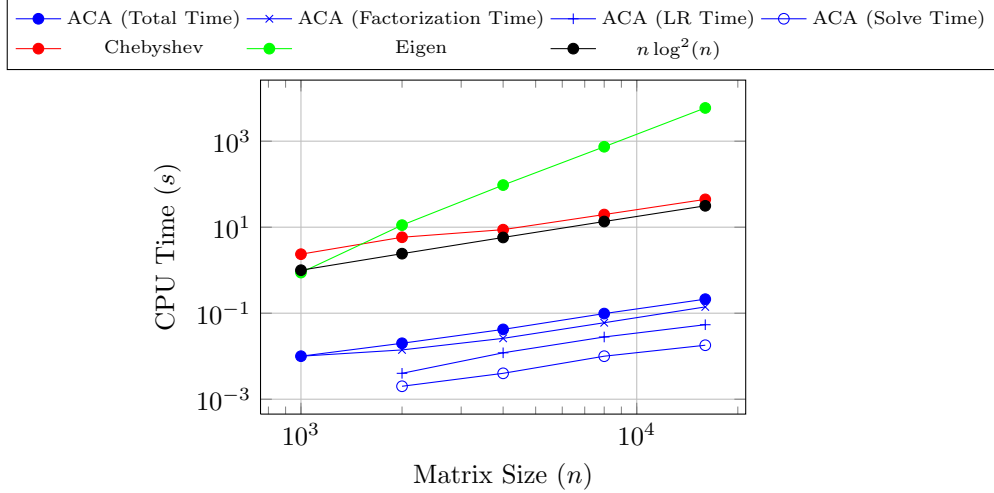


FIG. 6.3. CPU time (seconds) vs. matrix size (n) for matrices defined with the radial basis function kernel $(1 + d^2)^{-1/2}$. Similar plots were obtained for all the kernels shown in Fig. 6.2 and the trends are similar, in particular the asymptotic growth is $O(n \ln^2 n)$. This plot shows the CPU time for a variety of solve methods including Eigen’s colQR solver [16], the fast solver with Chebyshev low-rank approximation, and various stages of the fast solver with the ACA scheme. The diagonal entry of all matrices is set to zero. The ACA tolerance (ϵ_{ACA}) for the ACA solver was 10^{-10} and the rank of approximation for the Chebyshev solver was 30. For Chebyshev, the rank is fixed and therefore the computational time grows very closely like $O(n \ln^2 n)$. For ACA, the rank varies with n (and the accuracy ϵ) and therefore the asymptotic growth with n is more difficult to determine. However, based on these numerical benchmarks, we observe an $O(n \ln^2 n)$ growth with ACA as well. LR Time in the legend is the time required to calculate the low-rank factorization.

tolerance (ϵ_{ACA}). Further, this error decreases linearly with the decrease of the tolerance.

We have also compared the speed of our solver with a conventional linear algebra package solver, namely Eigen’s QR solver. Figure 6.4(b) shows the results of such comparison. As can be seen, our solver is roughly 10 times faster than Eigen’s QR solver.

6.3. FETI-DP Local Matrix Schur Complement Results. Figure 6.5 shows the results of applying our black-box HODLR solver to a 3D FETI-DP local matrix Schur complement. Here, SVD accuracy is the accuracy we used to obtain our low-rank approximations. As can be seen, our solve time is much faster compared to Eigen even for high accuracies. Further, we still have reasonable solution accuracy even at relatively low-rank approximation accuracies (10^{-3} solution accuracy at 10^{-1} SVD accuracy).

In this benchmark, we did not utilize a fast low-rank approximation scheme like the ACA or the pseudo-skeleton algorithm. Application of these algorithms requires tuning various algorithm parameters which we did not investigate. Our global for this benchmark was to show that Schur complements appearing in the elimination process of a three-dimensional sparse matrix corresponding to a finite-element problem with multiple degrees of freedom per node can be approximated as an HODLR matrix.

7. Conclusion and Future Work. We have developed a black-box fast direct solver for HODLR matrices. We used fast algebraic low-rank approximation schemes like the ACA algorithm, which enables us to apply our method to any HODLR matrix without any prior knowledge of the matrix kernel. The benchmarks show that our solver is both fast and accurate. Further, the results shown in this report serve as a proof of concept that the “frontal” matrices appearing in the multifrontal elimination scheme for a sparse matrix associated with a finite-element discretization of an elliptic PDE in both two and three dimensions, have an HODLR structure. Thus, they can be solved with reasonable accuracy and lot less computational cost.

An aspect that we have not described in this paper is the use of these fast direct solver methods as

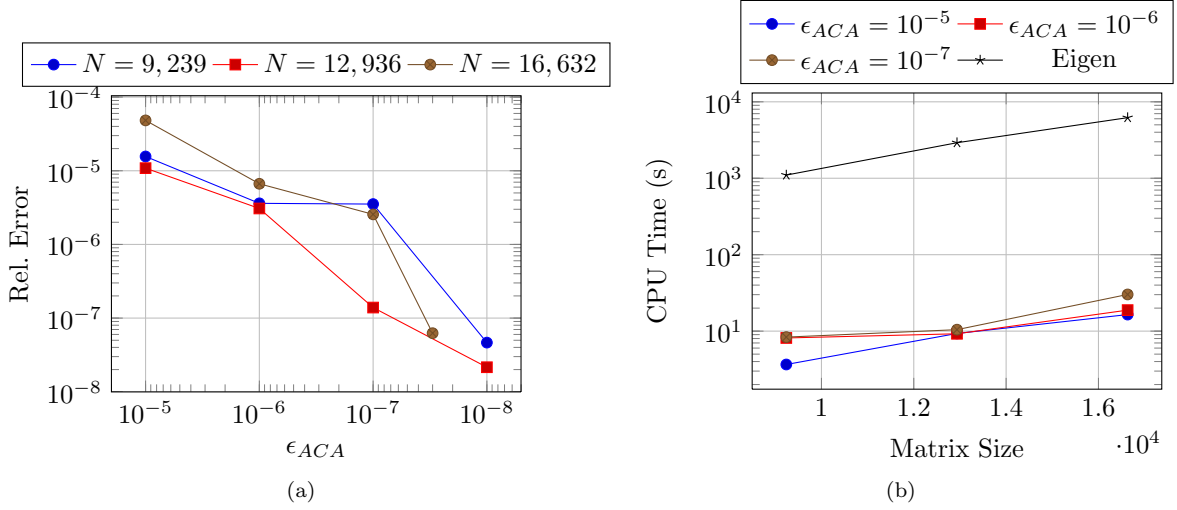


FIG. 6.4. Application of the HODLR solver to the turbine blade Schur complement. a) ACA solver relative error vs. ACA tolerance (ϵ_{ACA}) for various matrix sizes. b) ACA solver CPU time vs. matrix size for various ACA tolerances (ϵ_{ACA}).

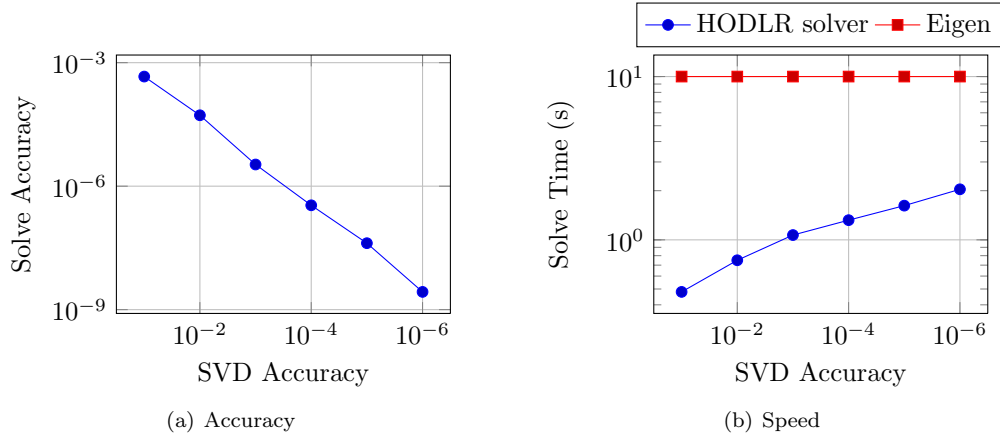


FIG. 6.5. Solve time and solve accuracy vs. SVD accuracy. Solve time consists of both factorization and solve time. It does not account for low-rank approximation time however.

preconditioners. In that case a low accuracy can be used for the direct solver. As soon as $\rho(I - A_{\text{HODLR}}^{-1}A) \ll 1$ (where A is the matrix, A_{HODLR}^{-1} corresponds to the approximate solve using the fast solver, ρ denotes the spectral radius), we can get convergence with a few iterations using any classical iterative scheme, e.g.:

$$x_{n+1} = x_n + A_{\text{HODLR}}^{-1}(b - Ax_n)$$

The HODLR scheme is a simple method to develop efficient direct solvers. Its advantages are the ease of implementation. It works very well for medium to large problems. It can be coupled with any type of low-rank compression scheme available. Its disadvantages is that it relies on off-diagonal blocks being low-rank which is not true asymptotically as $N \rightarrow \infty$ for points distributed on a 2D or 3D manifold. However, the HODLR scheme has a computational cost of $\mathcal{O}(n \log^2 n)$ with a small constant in front of $n \log^2 n$. As a result, even if the rank r increases, the method stays competitive. We estimated that the direct solver

we presented starts being faster as soon as the rank r is $r \sim 0.4n$ compared to an $(2/3)n^3$ LU factorization algorithm.

We note the work Ho and Ying [13, 14] where they developed a direct solver variant to overcome the growth of the rank. The idea is to use interpolative factorizations to build low-rank approximations. Then, as nodes tend to accumulate near the boundary of the manifold, the same interpolative factorization is reapplied to a different set of points to further reduce the rank of the approximation. This guarantees a solver with a computational cost of $O(n)$ in 2D or 3D. The downside of this approach is that it requires a knowledge of the spatial location of the degrees of freedoms in the matrix, an information that is not often readily available.

REFERENCES

- [1] S. AMBIKASARAN AND E. DARVE, *An $O(n \log n)$ fast direct solver for partial hierarchially semi-separable matrices*, Journal of Scientific Computing, (2013).
- [2] J.D. BENAMOU AND B. DESPRES, *A domain decomposition method for the Helmholtz equation and related optimal control problems*, Journal of Computational Physics, 136 (1997), pp. 68–82.
- [3] S. CHANDRASEKARAN, M. GU, X.S. LI, AND J. XIA, *Some fast algorithms for hierarchially semiseparable matrices*, tech. report, UCLA, 2008.
- [4] S. CHANDRASEKARAN, M. GU, AND T. PALS, *A fast ULV decomposition solver for hierarchially semiseparable representations*, SIAM J. Matrix Anal., 28 (2006), pp. 603–622.
- [5] I.S. DUFF AND J.K. REID, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Transactions on Mathematical Software, 9 (1983), pp. 302–325.
- [6] C. FARHAT, M. LESOINNE, P. LETALLEC, K. PIERSON, AND D. RIXEN, *FETI-DP: a dual-primal unified FETI method – Part 1: A faster alternative to the two-level FETI method*, International Journal for Numerical Methods in Engineering, 50 (2001), pp. 1523–1544.
- [7] W. FONG AND E. DARVE, *The black-box fast multipole method*, Journal of Computational Physics, 228 (2009), pp. 8712–8725.
- [8] A. GILLMAN, P.M. YOUNG, AND P.G. MARTINSSON, *A direct solver with $O(N)$ complexity for integral equations on one-dimensional domains*, Frontiers of Mathematics in China, 7 (2012), pp. 217–247.
- [9] S.A. GOREINOV, E.E. TYRTYSHNIKOV, AND N.L. ZAMARASHKIN, *A theory of pseudoskeleton approximations*, Linear Algebra and Its Applications, 261 (1997), pp. 1–21.
- [10] J. C. HAN, S. DUTTA, AND S. EKKAD, *Gas Turbine Heat Transfer and Cooling Technology*, CRC Press, second edition ed., 2013.
- [11] F. HECHT, *New development in freefem++*, Journal of Numerical Mathematics, 20 (2012), pp. 251–265.
- [12] K.L. HO AND L. GREENGARD, *A fast direct solver for structured linear systems by recursive skeletonization*, SIAM J. Sci. Comput., 34 (2012), pp. A2507–2532.
- [13] K. HO AND L. YING, *Hierarchical interpolative factorization for elliptic operators: differential equations*, arXiv preprint arXiv:1307.2895, (2013).
- [14] ———, *Hierarchical interpolative factorization for elliptic operators: differential equations*, arXiv preprint arXiv:1307.2895, (2013).
- [15] B.M. IRONS, *A frontal solution program for finite element analysis*, International Journal for Numerical Methods in Engineering, 2 (1970), pp. 5–32.
- [16] B. JACOB, *Eigen v3*. <http://eigen.tuxfamily.org>, 2010.
- [17] W.Y. KONG, J. BREMER, AND V. ROKHLIN, *An adaptive fast direct solver for boundary integral equations in two dimensions*, Applied and Computational Harmonic Analysis, 31 (2011), pp. 346–369.
- [18] J.W.H. LIU, *The multifrontal method for sparse matrix solution theory and practice*, SIAM Review, 34 (1992), pp. 82–109.
- [19] P.G. MARTINSSON, *A fast direct solver for a class of elliptic partial differential equations*, Journal of Scientific Computing, 38 (2009), pp. 316–330.
- [20] P.G. MARTINSSON AND V. ROKHLIN, *A fast direct solver for boundary integral equations in two dimensions*, Journal of Computational Physics, (2005), pp. 1–23.
- [21] F. PELLEGRINI AND J. ROMAN, *SCOTCH: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs.*, High-Performance Computing and Networking, 1067 (1996), pp. 493–498.
- [22] S. RJSANOW, *Adaptive cross approximation of dense matrices*, in IABEM, International Association for Boundary Element Methods, 2002.
- [23] P.G. SCHMITZ AND L. YING, *A fast direct solver for elliptic problems on general meshes in 2D*, Journal of Computational Physics, 231 (2012), pp. 1314–1338.
- [24] D.G. WILSON AND T. KORAKIANITIS, *The Design of High-Efficiency Turbomachinery and Gas Turbines*, Prentice Hall, second edition ed., 1998.
- [25] F. WOOLFE, E. LIBERTY, V. ROKHLIN, AND M. TYGERT, *A fast randomized algorithm for the approximation of matrices*,

- Applied and Computational Harmonic Analysis, 25 (2008).
- [26] J. XIA, S. CHANDRASEKARAN, M. GU, AND X. S. LI, *Superfast multifrontal method for large structured linear systems of equations*, SIAM J. Matrix Anal., 31 (2009), pp. 1382–1411.
- [27] J. XIA, S. CHANDRASEKARAN, M. GU, AND X. S. LI, *Fast algorithms for hierarchially semiseparable matrices*, Numerical Linear Algebra with Applications, 17 (2010), pp. 953–976.